Optimization of Custom MOS Circuits by Transistor Sizing

Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, Chandu Visweswariah
IBM T. J. Watson Research Center and IBM Microelectronics Division
Yorktown Heights, NY, Austin, TX and Burlington, VT

Abstract

Optimization of a circuit by transistor sizing is often a slow, tedious and iterative manual process which relies on designer intuition. Circuit simulation is carried out in the inner loop of this tuning procedure. Automating the transistor sizing process is an important step towards being able to rapidly design high-performance, custom circuits. JiffyTune is a new circuit optimization tool that automates the tuning task. Delay, rise/fall time, area and power targets are accommodated. Each (weighted) target can be either a constraint or an objective function. Minimax optimization is supported. Transistors can be ratioed and similar structures grouped to ensure regular layouts. Bounds on transistor widths are supported.

JiffyTune uses LANCELOT, a large-scale nonlinear optimization package with an augmented Lagrangian formulation. Simple bounds are handled explicitly and trust region methods are applied to minimize a composite objective function. In the inner loop of the optimization, the fast circuit simulator SPECS is used to evaluate the circuit. SPECS is unique in its ability to efficiently provide time-domain sensitivities, thereby enabling gradient-based optimization. Both the adjoint and direct methods of sensitivity computation have been implemented in SPECS.

To assist the user, interfaces in the Cadence and SLED design systems have been constructed. These interfaces automate the specification of the optimization task, the running of the optimizer and the back-annotation of the results on to the circuit schematic.

JiffyTune has been used to tune over 100 circuits for a custom, high-performance microprocessor that makes use of dynamic logic circuits. Circuits with over 250 tunable transistors have been successfully optimized. Automatic circuit tuning has been found to facilitate design re-use. The designers' focus shifts from solving the optimization problem to specifying it correctly and completely. This paper describes the algorithms of JiffyTune, the environment in which it is used and presents a case study of the application of JiffyTune to individual circuits of the microprocessor.

1. Introduction, motivation and previous work

Designers often spend a lot of time manually sizing their schematics for area, delay and power, particularly in the context of custom designs. The tuning process is iterative, slow, tedious and error-prone, with circuit simulation in the inner loop. The updating of transistor widths from one iteration to the next relies on human intuition. Automating the circuit optimization process is an important step towards rapidly designing high performance, custom circuits. Automatic circuit tuning has the additional benefit of facilitating design adaptation and re-use. Hence an automatic tuning (and retuning) capability is crucial to the productive design of custom circuits.

There have been many attempts to automate the transistor sizing problem. The first class of methods [1, 2] is based on static timing analysis [3] in which the circuit is assumed to consist of pre-characterized library cells. The delay of each cell is available as an analytic function of the sizes of the transistors in the cell. Total path delay is expressed as a function of the individual transistor widths and optimized. In particular, if the Elmore delay model [4,5] is used, this overall delay is seen to be a posynomial function (a particular algebraic form) of the transistor widths. By a simple mapping of variables, the objective is converted to a convex function [1], and hence any minimum of the latter is guaranteed to be a global minimum. The advantages of static-timing-based methods include efficiency, ability to handle large designs and freedom from requiring input patterns to carry out the tuning. One of the problems with these methods is that they are not applicable to full-custom circuit designs, since static timing analyzers usually rely on pre-characterized library cells. Second, the accuracy of static timing analysis is limited (to about $\pm 25\%$ in our experience) making it unsuitable as a basis for tuning high-performance custom circuits. Finally, static timing analysis is prone to the *false path problem*, so the optimizer may be working hard to tune paths that are either irrelevant or can never be sensitized. Recently, power optimization has been proposed in this general framework [6]. Power is measured by probabilistic methods [7] and then approximated by a posynomial function. Simultaneous tuning of drivers and interconnect has been proposed in [8, 9].

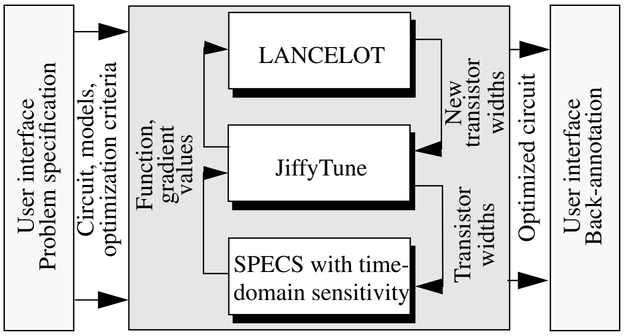
Tuning based on *dynamic simulation* overcomes many of the above limitations of static tuning. The accuracy is as good as the simulator employed, false paths are not a problem and the method is applicable to any custom circuitry that the simulator can analyze. Appropriate input patterns must be provided by the user. These methods [10, 11] typically run SPICE in the inner loop to optimize such circuit performance functions as gain, area, delay and phase margin. However, using SPICE iteratively is computationally expensive and limits the size of the circuit that can be tuned. From an overall design perspective, we see static and dynamic methods complementing each other at different stages of the methodology, depending on the type of design.

In this paper, we present a method for tuning custom MOS circuits that uses dynamic simulation and *gradient-based optimization*. Our ability to compute gradients efficiently is crucial to the success of this approach. JiffyTune is a prototype implementation of our method. An overview of JiffyTune is presented in Section 2. JiffyTune uses SPECS [12, 13], a fast circuit simulator, to evaluate the circuit and provide function and gradient values. SPECS and the computation of sensitivities are the topics of Section 3. The optimization engine used in JiffyTune is LANCELOT [14, 15, 16], a large-scale nonlinear programming package that handles simple bounds explicitly and accommodates general constraints with an augmented Lagrangian formulation. LANCELOT has been customized to the circuit tuning problem. The numerical methods involved in the nonlinear optimization are described in Section 4. To make the tuning environment productive and intuitive, interfaces have been built in two different design environments. Section 5 is devoted to the concepts guiding these interfaces and their benefits. JiffyTune has been used on many custom, dynamic-logic circuits of a high-performance microprocessor. A case study of the application of JiffyTune to this chip design, along with benchmarks, is presented in Section 6, followed by a section containing conclusions and future work.

ICCAD '96
1063-6757/96 \$5.00 © 1996 IEEE

2. Overview of JiffyTune

This section provides an overview of the various high-level software components of JiffyTune, as depicted in Figure 1. Subsequent sections contain detailed descriptions of the individual components.



JiffyTune "Engine"

Figure 1. High-level view of JiffyTune.

The JiffyTune "engine" solves the following problem. Given a circuit schematic, input signals, a list of tunable transistors with initial widths and a set of circuit performance requirements, determine the optimal assignment of transistor widths to tunable transistors in order to achieve the requirements. The user interface makes it convenient for the user to specify the problem, and visualize and accept the results of optimization.

2.1. JiffyTune

The JiffyTune block in Figure 1 performs the administrative portion of the tuning task. A control file grammar has been defined for the specification of circuit optimization problems. The control file contains the following information.

Parameters: This section contains a list of tunable transistors, their initial widths and bounds. Tunable transistors can be ratioed to other tunable transistors. Further, the user interface allows *grouping* of instances of similar structures so that they track each other during tuning. Thus, for example, the cells of an n -bit wide multiplexer can be grouped to ensure that the cells stay identical through the tuning process and thus lend themselves to a structured, regular layout.

Measurements and functions: A measurement is either a crossing-time, power or area measurement. In the absence of layout information, area is modeled by the sum of the tunable transistors' widths. *Functions* consist of any linear combination of measurements. Thus delays and rise/fall times are typically the difference of two crossing times. Each function has a weight, a target and a relation. A relation of "less than" implies that this function should be less than the target value. Similarly, relations of "greater than" and "equal to" are allowed. Alternately, a function can be "minimized" which means that the optimizer will try to decrease the value of this function as much as possible. Weights can be used to explore various trade-offs in tuning the circuit; they are especially required when functions of different quantities (area, delay, power) are being combined into a composite objective function.

Any number of functions can be grouped into a *minimax function*. A minimax function implies that the largest of some number of functions needs to be minimized or must meet a constraint. For example, the statement of the problem might be to minimize the delay of the worst of three paths through some combinational logic block.

Controls: This section provides administrative information like the maximum number of iterations, the layout grid for rounding transistor widths at the end of optimization and the location of the device model files.

JiffyTune reads the control file and internally represents the problem in a format that is understood by LANCELOT. JiffyTune also provides to LANCELOT a callable routine that will accept a set of transistor widths, perform a SPECS simulation, and return function and gradient values in the form required by LANCELOT. Then JiffyTune begins a LANCELOT optimization. At each iteration, JiffyTune keeps track of the best results so far. One of the main functions of the JiffyTune block is to *chain rule and combine* gradients to provide to LANCELOT the gradients of various functions with respect to *independent variables only.* Typically, 25 to 30 iterations are required for convergence. The default maximum number of iterations in JiffyTune is 50. The recently implemented *slack updating method* described in Section 4.2 has led to fewer iterations being required in general.

2.2. SPECS

SPECS is a fast circuit simulator that uses simplified device models and event-driven techniques. JiffyTune calls SPECS in the inner loop to evaluate the circuit, and provide function and gradient values. SPECS and its sensitivity computation capabilities are described in Section 3.

2.3. LANCELOT

LANCELOT is a large-scale nonlinear optimization package that handles simple bounds and general constraints. JiffyTune provides the problem description and initial transistor sizes to LANCELOT. LANCELOT repeatedly calls SPECS with different transistor size settings, and builds a model of the "performance surface" of the circuit. It uses sophisticated nonlinear programming techniques to minimize the objective function. Details regarding LANCELOT and its application to the circuit tuning problem are provided in Section 4.

In addition to LANCELOT, the Levenberg-Marquardt [17] and Minos [18] optimization packages have been integrated into JiffyTune. The optimization testing environment described in [19] was used to integrate Minos into JiffyTune. The Levenberg-Marquardt method is limited since it only performs unconstrained optimization and is relatively unsophisticated. The Minos integration has been used only for comparisons and "sanity checks."

2.4. The user interface

JiffyTune requires a knowledgeable user to carefully specify the optimization problem, and greedily takes advantage of any unspecified aspects. Further, the engine requires a control file that is difficult to create manually, particularly for large circuits. The user interface helps the user concentrate on the specification of the optimization problem by providing an intuitive interface and eliminating the tedium of dealing with a file-driven tool. It also provides facilities for back-annotation of the results of tuning. Section 5 is devoted to a discussion of the environment in which JiffyTune is used and the description of the interface.

3. SPECS and time-domain sensitivity computation

SPECS (Simulation Program for Electronic Circuits and Systems) is a fast circuit simulation program. SPECS is on the average 70x faster than AS/X, an internal traditional circuit analysis program [20] like SPICE. SPECS uses simplified device models and event-driven techniques to efficiently simulate MOSFET circuits in the time-domain, and has been used in production mode in various integrated circuit designs. JiffyTune uses SPECS to evaluate the circuit being optimized. However, this paper will not describe SPECS in any detail. The reader is referred to [12, 13, 21]. The device modeling assumptions in SPECS restrict its relative timing accuracy to $\pm 5\%$, and hence JiffyTune can only tune to within this accuracy limit.

3.1. Sensitivity computation

SPECS uses simplified device models that consist of piecewise constant $i - v$ characteristics in multiple dimensions and grounded, linear capacitances. These simplifications allow efficient, incremental time-domain sensitivity computation [22, 23, 24]. Both the adjoint [25, 21] and direct [26] method have been implemented. In the direct method, branch constitutive relations (device characteristics) are *directly differentiated* with respect to the sensitivity parameter of interest. The circuit reflecting these differentiated equations, called the *sensitivity circuit*, has the same topology as the original circuit. Since SPECS uses piecewise constant device models, the sensitivity circuit consists of *disconnected capacitances* for large sub-intervals of time, with occasional *impulses of currents* flowing between these capacitances at times corresponding to events in the nominal simulation. Thus the solution of the sensitivity circuit is extremely efficient. In the direct method, the sensitivities of *all functions* with respect to *one parameter* are computed with a single solution of the sensitivity circuit.

In the adjoint method, elements are replaced by *adjoint equivalents* based on Tellegen's theorem [25, 21]. Again, the circuit is very simple and lends itself to efficient solution. In this case, however, *time is run backwards* in the adjoint circuit, and the waveforms of the adjoint circuit are *convolved* with those of the original circuit to obtain the required sensitivities. The gradients of *one function* with respect to *all parameters* are computed in a single solution of the adjoint circuit. Hence, when there are sufficiently more parameters than functions to justify the overhead of convolution, the adjoint method is advantageous.

Once the approximation in the simplified device models is accepted, the computation of gradients is exact. After the sensitivity circuit is solved in either method, gradients are chain-ruled and combined to obtain the sensitivity of each function with respect to *all the ramifications of variation of the tunable transistors' widths*. When the width of a transistor varies, its source and drain diffusion capacitance and all the intrinsic MOSFET parasitic capacitances change. Each of these is submitted as an internal sensitivity parameter and then all the gradients are postprocessed and combined appropriately. The flavor of these computations is captured by the following simplified equation.

$$\begin{aligned}\frac{df}{dW} &= \frac{\partial f}{\partial W_{eff}} \cdot \frac{dW_{eff}}{dW} + \frac{\partial f}{\partial CD_{total}} \cdot \frac{dCD_{total}}{dW} \\ &\quad + \frac{\partial f}{\partial CS_{total}} \cdot \frac{dCS_{total}}{dW} + \frac{\partial f}{\partial CG_{total}} \cdot \frac{dCG_{total}}{dW}\end{aligned} \tag{1}$$

where f is the sensitivity function of interest, W is the transistor width (sensitivity parameter), W_{eff} is the effective width and CG_{total}, CS_{total} and CD_{total} are the total parasitic capacitance at the gate, source and drain nodes, respectively. (1) is further expanded in terms of the device model parameters.

Voltage crossing sensitivities are expressed in terms of the nominal voltage waveform and transient sensitivity waveform of the appropriate signal, both sampled at the time of the voltage crossing of interest. In the case of the adjoint method, the transient sensitivity waveform is sampled by expressing the required value as a convolution integral and choosing to excite the adjoint circuit by an appropriate current source connected to that node.

3.2. Sensitivity benchmarks

The number of time-domain gradients computed during a typical JiffyTune run may be in the millions! Hence gradient computation must be extremely efficient to make this process feasible. A dynamic logic "branch scan" circuit with 144 MOSFETs was chosen to demonstrate the efficiency of gradient computation. The circuit was simulated in SPECS for a simulation interval of 27 ns. The CPU time for simulation was 2.05 s on an IBM Risc/System 6000 model 590. Then the same simulation run was carried out with 36 sensitivity functions (crossing times) and 104 MOS transistor widths as sensitivity parameters. Since there were 64 diffusion and other parasitic capacitances dependent on these 104 transistor widths, the total number of sensitivity parameters was 168. The number of gradients computed in this benchmark was 6,048, since SPECS finds the gradient of *every* sensitivity function with respect to *each* sensitivity parameter (our Jacobian matrix is dense). The number of sensitivities required was unusually large in this example, which was chosen to showcase the efficiency of gradient computation. The run times of SPECS with both the adjoint and direct method on this benchmark circuit are shown in Table 1. From the table, we see that the total run time for a JiffyTune iteration would be 24.94 s (assuming that the direct method were used). For comparison, the AS/X [20] run time on this circuit (with no gradient computation, of course) was 40.11 s. Hence, even on this modest example, JiffyTune can almost complete two iterations *with gradient computation* in the time it takes AS/X to simulate the nominal circuit once.

Table 1: Sensitivity computation run time.

Run time in CPU seconds	Adjoint method	Direct method
Total run time	32.38	24.94
Run time for sensitivity computation only	30.32	22.89
Run time per sensitivity circuit solution	0.84	0.14
Run time per sensitivity circuit solution as a fraction of simulation time (2.05 s)	40.78%	6.65%
Run time per gradient computation	5.01e-3	3.78e-3
Run time per gradient evaluation as a fraction of simulation time	0.24%	0.18%

As can be seen from the table, the overhead of computing *one gradient* is a fraction of a percent of the original simulation time, which works out to 5 ms or less of CPU time in this example! The overhead of *one sensitivity circuit analysis* is about 7% for the direct method and about 40% for the adjoint method. Note that the number of runs in the adjoint method is equal to the number of functions, while it is equal to the number of sensitivity parameters in the direct method. The higher overhead in the adjoint method is accounted for by the convolution required between the waveforms of the original circuit and the sensitivity circuit. SPECS inspects the number of functions and the number of parameters and automatically makes a judgment, based on a simple heuristic, as to which method will be more efficient. The heuristic favors the adjoint method if the number of parameters exceeds the number of functions by a factor of 5. This heuristic appears to be effective in practice.

4. Nonlinear optimization in JiffyTune

4.1. LANCELOT

The optimization engine of JiffyTune is based upon the large-scale nonlinear programming package LANCELOT. The kernel algorithm is an adaptation of a trust region method to the general nonlinear optimization problem subject to simple bounds. The method is extended to accommodate general constraints by using an augmented Lagrangian formulation and the bounds are handled directly and explicitly via projections that are easy to compute.

In the context of *unconstrained optimization*, trust region methods, combining an intuitive framework with a powerful and elegant theoretical foundation, have led to robust numerical implementations. An excellent reference is [27]. The basic idea of trust region methods is to approximately minimize a model of the objective function in a local neighborhood (called the *trust region*) centered at the current point. The objective function is modeled about the current point $x^k \in \Re^n$, where k is the iteration count and x is the n-vector of variables. To minimize the model in the trust region, a step s^k is taken at iteration k to arrive at the point $x^k + s^k$. The function is evaluated at this point to determine how well the model predicted the actual change in the objective function. If good descent is obtained, the approximate minimizer is accepted as the next iterate $\left(x^{k+1} \leftarrow x^k + s^k\right)$ and the trust region is expanded. If moderate descent is obtained, the trust region size remains unchanged, but the step is accepted. Otherwise, no new point is accepted and the trust region is contracted. The beauty of such an approach is that, when the trust region is small enough and the problem smooth, the approximation is good, provided the gradients are sufficiently accurate. Moreover, assuming one does at least as well as the minimum along the steepest descent direction of the model within the trust region (which determines the so-called Cauchy point), one can ensure convergence to a stationary point. In addition, the trust region is eventually expanded so that it does not interfere with the subsequent iterates, and thus, assuming that in this situation the underlying algorithm is sufficiently sophisticated, one can ensure fast asymptotic convergence.

The extension of the above ideas to *problems with simple bounds* is relatively straightforward and is illustrated in Figure 2, for a quadratic model function and an l_∞ trust region. Essentially, one generalizes the Cauchy point (v_k in the figure) to the minimum along the *projected* gradient path ($x^k - u - w$) within the trust region, where the projection is with respect to the bounds (either those provided by the user or implicit in the trust region). As in the previous case, global convergence can be guaranteed, provided one does at least as well as the generalized Cauchy point (w). If a variable, as determined by the generalized Cauchy point, is at a bound it is said to be an *activity.* Unbounded variables are *free*. Activities are fixed temporarily, thus reducing the dimensionality of the search space (from two to one in the figure). Then, using only the free variables, the model of the objective function is further minimized within the feasible region and within the trust region (w is optimal in Figure 2). Thus one obtains better convergence, and ultimately, satisfactory asymptotic convergence. Updating of the trust region size and current point is handled in exactly the same way as it is in the unconstrained case.

It has been proved [14] that this method converges to a Kuhn-Tucker point [29]. Moreover, the correct active simple bounds are

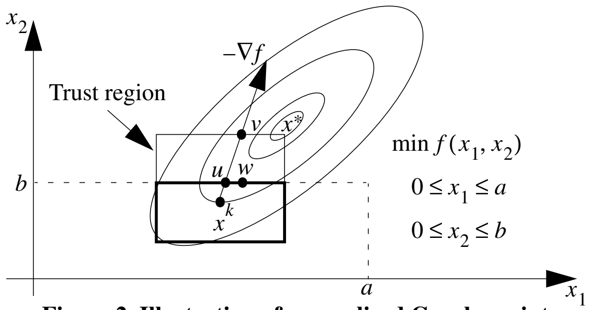


Figure 2. Illustration of generalized Cauchy point.

identified after a finite number of iterations assuming that strict complementarity is satisfied and the activities determined by the generalized Cauchy point are kept active during the rest of the iteration when the model is further reduced. Details are given in [14].

The *extension to handle equality constraints* is carried out by means of an augmented Lagrangian function

$$\Phi(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i=1}^{m} c_i(x)^2 . \qquad (2)$$

Φ is minimized subject to the explicit bounds, using the earlier algorithm. Here f is the objective function, x the variables of the optimization, $c_i(x)$ is an equality constraint with λ_i being the corresponding Lagrangian multiplier and μ the penalty parameter used to dynamically weight feasibility. Inequality constraints are converted to equality constraints by first introducing slack or surplus variables, if necessary, and then formulating the augmented Lagrangian as before. This approach can be summarized as follows:

1. Test for convergence using the two following conditions. *Sufficient stationarity* -- the projected gradient of the augmented Lagrangian with respect to the simple bounds is sufficiently small, and *sufficient feasibility* -- the norm of the constraint violations is sufficiently small.
2. Use the simple bounds algorithm to find an approximate stationary point (minimizer) of Φ subject to simple bounds.
3. If sufficiently feasible, update the multipliers λ_i and decrease the tolerances for stationarity and feasibility.
4. Otherwise, give more weight to feasibility (decrease μ) and reset tolerances for stationarity and feasibility.

It is possible to show, under suitable conditions, that convergence to a first-order stationary point for the nonlinear programming problem is attained. Further, if there is a *single limit point*, eventually the penalty parameter μ is not reduced. Details of these and other theoretical properties are given in [15] and [28].

A significant cost in the optimization is solving a linear system of equations. Typically these arise from the necessity to determine an approximate stationary point for a quadratic function -- equivalently, the necessity to solve a linear system whose coefficient matrix is symmetric. If the system is large, there are two approaches. The first is to use direct methods based on multifrontal techniques (see Chapter 10 of [30]). Our experience to date, however, has been that an iterative approach using preconditioned conjugate gradients is more robust. All our reported numerical results with JiffyTune use this method. The appeal of conjugate-gradient methods for large-scale optimization is that they are particularly simple and only require that we store a few vectors. Moreover, they can be significantly accelerated by the use of preconditioners. Perhaps the best known conjugate basis for a convex quadratic form is the set of eigenvectors of the Hessian. The essential result is that at each iteration, the conjugate gradient method *minimizes the quadratic model* in the space spanned by the corresponding conjugate basis. If we can cluster eigenvalues (i.e., approximately have multiple eigenvalues) we can reduce the number of iterations for good approximations to minimizers from close to n to close to the number of clusters. The perfect way to do this in the quadratic case is to precondition with the Hessian inverse -- but then this is equivalent to carrying out Newton's method. Surprisingly, one can often do very well by using crude approximations to the Hessian (diagonal matrices, for instance). A good description of conjugate methods is given in [29], Sections 4.8.3 and 4.8.5. The LANCELOT package offers several preconditioners, and the Schnabel-Eskow preconditioner [31] is used in JiffyTune. A detailed reference on the LANCELOT package, including all the available options, is given in the book [16] that accompanies the original software.

4.2. Application of LANCELOT to JiffyTune

In the context of JiffyTune it was necessary to make certain modifications to LANCELOT to account for the fact that the function and gradient values from SPECS, although accurate to within small perturbations, are noisy. The introduced errors are small but significantly larger than machine precision. Because of the complexity of general nonlinear optimization, many initializations (such as the choice of the trust region radius or quadratic model) are based upon intelligent guesses, which cannot, of course, be ideal in all circumstances. In the worst case, for functions without noise, unfortunate choices can result in inefficiencies, but in the noisy case they can be insurmountable. A trivial example is if movement of less than 0.001 microns in transistor width is considered negligible, it may be disastrous if an automatic choice of the initial trust region size produces a radius that is much smaller. For similar reasons, we had to introduce looser tolerances for feasibility, line search discontinuities and bound activities, which are based upon machine precision in the original software. Finally, in order to stop gracefully and predictably we needed to consider step sizes beneath which further progress is unlikely and relate stopping criteria to this step size in a robust and consistent manner.

Two other enhancements deserve special mention. Slack/surplus variables corresponding to satisfied inequalities are updated at each iteration so that the corresponding equality is satisfied exactly, whenever such an update is consistent with the convergence theory; the result has been a reduction in the number of iterations to convergence.

Minimax optimization is handled by the introduction of an additional linear variable and reformulating the problem as a general nonlinear programming problem. For example, suppose one had the problem

$$\underset{x \in \Re^n}{\text{minimize}} \quad \underset{i \in M \equiv \{1, 2, \ldots, m\}}{\text{maximum}} \quad f_i(x) \quad . \tag{3}$$

This problem can be reformulated as

$$\underset{z \in \Re,\, x \in \Re^n}{\text{minimize}} \quad z \tag{4}$$

subject to the inequality constraints $z - f_i(x) \geq 0, \quad 1 \leq i \leq m$.

5. JiffyTune interface and environment

The JiffyTune engine as described above is driven by a textual control file that describes the optimization problem. Manual preparation and editing of such a file is tedious and error-prone. Also, the sophistication of LANCELOT and the choices of algorithms and tolerances thereof are not directly relevant to the end user. Thus, from the inception of the JiffyTune project, it was realized that a good human interface and an intuitive abstraction of its use and behavior would be crucial to acceptance of the tool by circuit designers. Interfaces were built to run the tool from the Cadence [32] and SLED [33] schematic design systems. The interface in the Cadence design environment was evolved simultaneously with the JiffyTune engine. Integrating the tool into such a framework capitalizes on the familiarity of the user with the schematic design environment, and lends a visual and interactive aspect to the tool. Many of the complexities are hidden from the designer, although care was taken to allow full access to all tool functions, if the designer so requires. The basic functions of the interface are listed below.

Specification of tuning parameters: Tunable transistors are specified simply by selecting transistors or gates on a hierarchical schematic. The tunable transistors/gates are visually marked by a flag to indicate tunability. Facilities are provided to ratio transistors. Thus the two NFETs in a NAND gate can be forced to have the same width or adhere to a given tapering ratio. In addition, similar instances (transistors, gates or higher-level functional blocks) can be "grouped" together, to ensure that corresponding transistors in those blocks track during tuning.

Specification of measurements and functions: Presently, the interface supports delay, transition time (slew), area and power functions. For delay and transition times, net selection is done directly on the schematic. Power functions are specified by selecting the required voltage source, again directly on the schematic. In all cases, the user is prompted to provide a relation and target value as described in Section 2.1. In the schematic environment, with no knowledge of layout, area targets are approximated by the sum of the widths of the tunable transistors. The appropriate linear combination of measurements is written to the control file in each case. Minimax functions can be defined over any set of existing measurements.

Specification of controls: Administrative information such as the maximum number of iterations, file location of device models and layout grid for rounding transistor widths at the end of optimization can be specified in a form that is pre-filled with project-specific defaults.

Execution of JiffyTune: After specifying parameters, functions and controls, the designer can ask for all this information to be written to a control file, which can be inspected or edited if required. Then the designer can launch the JiffyTune engine, whereupon the progress of the optimization is displayed.

Back-annotation of the results: The results of a JiffyTune run are back-annotated onto the schematic as *suggested transistor widths* next to the transistors (or as new parameters next to gates). The designer can then accept these new widths/parameters, selectively or as a whole. Further, a facility is provided to back-annotate final waveform characteristics, such as delay through a gate or rise time of a net, directly onto the schematics, relieving the designer of the need to browse through simulation data using a waveform viewer.

Utilities: As a courtesy to the designer, the JiffyTune menu also includes facilities replicated from other areas of the schematic design environment, such as schematic checking, netlisting and automatically adjusting the number of fingers on each transistor, to create a single integrated tuning environment. Portability to various different sites and projects has been achieved by carefully separating the main code of the user interface from configurable site- and project-specific code.

Circuit requirements must be specified with care, since the optimizer will take advantage of any unspecified aspects. For example, area minimization will shrink to its minimum size a transistor that does not contribute materially to any measured transition. Thus, the tool enforces clear expression of circuit requirements that otherwise are often tacit. Since these circuit requirements and attributes logically belong with the circuit (they are indeed part of the intellectual effort of designing the circuit), the tuning parameters and functions are stored in the design database, either as instance properties (tunability, upper and lower bounds on transistor widths) or as schematic properties (grouping, functions). This practice also encourages the reuse of circuits; if a circuit has been adequately specified, it can easily be retuned.

6. Case study of JiffyTune use

JiffyTune was applied to tune custom circuits in the critical paths of a high-performance, dynamic-logic microprocessor. The circuits consisted of a mix of transistors and continuously parameterized gates. Jiffytune made it possible to refine the transistor sizes of circuits more quickly and thus rapidly respond to design changes late in the chip design cycle. Thus more flexibility was preserved in changing the specifications of circuits. The Cadence graphical user interface made it possible for designers to use the tool with little or no training.

JiffyTune was used by 41 designers during about 1,200 interactive sessions to tune 168 unique circuits. Over 2,200 successful JiffyTune runs were carried out, showing that some circuits were re-tuned multiple times. The results of tuning on one particular benchmark circuit are presented below.

Table 2 lists the results of running JiffyTune on a 12-way priority decode circuit under four different conditions. The circuit contains 70 MOSFETs and the simulation was run for 35 ns. The tuning runs all had 64 tunable transistors, of which 16 were independent and 48 dependent. The 17 functions to be optimized included the rising delay through four critical paths, the falling delay through those paths, the rise/fall times on each of the above 8 transitions and an area constraint. For confidentiality purposes, the delay requirement on the worst of the critical paths has been normalized to 500 time units in our report on this benchmark. The table lists the rising and falling delay of the four paths being tuned *as predicted by AS/X* on the final design (the worst of the 8 delays for each run is shown in bold), the total tunable transistor area of the circuit and the CPU time required to run JiffyTune on an IBM Risc/System 6000 model 590. The first JiffyTune run (HOT) started from a circuit that had previously been manually tuned (Manual). The worst delay through the circuit improved by 7.5% and the area decreased by 5.0%. The second JiffyTune run (COLD) started from an untuned circuit in which initial transistor sizes were set to the same default value as they would be for a “new design.” Comparing the results of (HOT) and (COLD) shows that the poor start point did not change the final results, but the optimizer had to work harder. The next JiffyTune run (DELAY) was set up to cause JiffyTune to reach the timing goal of 500 time units at all cost. JiffyTune was configured as in run (HOT), only with a weight on the area constraint that was a tenth of the previous value. The table shows that the goal was reached but at a high cost in transistor area. In general, we have found that it is important to impose an area constraint. Without an area constraint, JiffyTune converges to one of many equally fast circuits depending on the start point, with some solutions more efficient in area than others. The final run used the same start point and weights as HOT, but formulated the problem as a minimax optimization. A solution with a slightly higher delay but lower area was obtained in this case.

Table 2: JiffyTune results for 12-way priority decode circuit; all delays are normalized to a requirement of 500 time units.

	Manual	HOT	COLD	DELAY	MINI-MAX
Path #1, falling delay	555	494	488	483	497
Path #1, rising delay	471	475	473	469	510
Path #2, falling delay	535	495	495	483	506
Path #2, rising delay	494	488	488	472	524
Path #3, falling delay	**561**	**519**	517	**497**	**544**
Path #3, rising delay	497	**519**	**519**	**497**	527
Path #4, falling delay	497	494	491	484	516
Path #4, rising delay	462	497	496	485	476
Area	893	844	849	1148	800
# JiffyTune iterations	--	9	26	16	41
Run time (CPU s)	--	172	465	289	716

JiffyTune in its present form is not directly applicable to designs in which gates are chosen from a fixed library of cells with a finite set of discrete power levels. JiffyTune performs well on hierarchical schematics with leaf cells containing any mix of transistors and continuously parameterized gates. In practice, JiffyTune handles circuits containing pass transistors well, in contrast to optimizers based on static timing analysis, since SPECS yields electrically true sensitivities taking into account details of the device model such as body effect. As new custom circuits are designed, JiffyTune will make it possible to speed up the design process, make more refined designs and provide better information about performance trade-offs.

7. Conclusions and future work

In this paper we described JiffyTune, a program that optimizes circuits by adjusting transistor sizes. JiffyTune makes use of fast simulation and time-domain gradient computation in the circuit simulator SPECS, and advanced nonlinear numerical techniques in the optimization package LANCELOT. Delay, rise/fall time, area and power optimization have been implemented. The optimization system is flexible and allows ratioing of transistors and grouping of identical instances. An intuitive interface including back-annotation of optimization results on to the schematic has been developed.

The environment in which a circuit will be used and the required performance are estimated long before the chip is built. By the time the circuit is integrated onto the chip, it may no longer be optimally tuned, much to the frustration of the design engineer. Changes in loading, changes in the specifications, changes in parasitics after extraction, changes in technology device models and remapping to a new technology are common occurrences during the course of a project. In such situations, retuning at the push of a button without tedious re-specification is extremely useful.

JiffyTune has been successfully used to tune a number of circuits on the critical paths of a high-performance microprocessor chip which makes liberal use of dynamic logic. It has been particularly useful in tuning tricky pass-gate circuits and has been found to enhance design re-use. Further, since the optimization process has been made easy and automatic for the designer, a paradigm shift has been observed; the issue becomes how to correctly specify the optimization problem rather than solving the optimization problem itself.

There are a number of avenues for future work. “Event-driven convolution” is expected to speed up the computation of gradients by the adjoint method in SPECS. Repeated solution runs of the sensitivity or adjoint circuit are independent and therefore amenable to parallel processing. Occasionally, we encounter “non-working circuits” in the course of the optimization, when a transition to be measured does not occur; recovery from such situations is an interesting problem. Extension to semi-infinite constraints [10] would allow optimization of circuits while taking into account environment variations such as temperature and power supply voltage. Reformulating the problem to take advantage of *group partial separability* in LANCELOT [16, 34] would speed up the optimization. If the optimization could be formulated as a mixed integer/continuous problem, transistor ordering could be part of the optimization procedure. In addition, applications to IC manufacturability are being considered.

8. Acknowledgments

The authors would like to thank I. Elfadel, E. Chiprout, D. Brand and S. Nassif for their suggestions and careful reading of the manuscript.

9. Bibliography

[1] J. P. Fishburn and A. E. Dunlop, “A posynomial programming approach to transistor sizing,” *IEEE International Conference on Computer-Aided Design,* pp. 326-328, November 1985.

[2] D. Marple, “Transistor size optimization in the Tailor layout system,” In Proceedings of the *ACM/IEEE Design Automation Conference,* pp. 43-48, June 1989.

[3] R. B. Hitchcock, Sr., G. L. Smith and D. D. Cheng, “Timing analysis of computer hardware,” *IBM Journal of Research and Development,* pp. 100-105, January 1982.

[4] W. C. Elmore, “The transient analysis of damped linear networks with particular regard to wideband amplifiers,” *Journal of Applied Physics,* volume 19, number 1, 1948.

[5] P. Penfield and J. Rubinstein, “Signal delay in RC tree networks,” *Proceedings of the 2nd Caltech VLSI Conference,* pp. 269-283, March 1981.

[6] S. S. Sapatnekar and W. Chuang, “Power vs. delay in gate sizing: conflicting objectives?,” *IEEE International Conference on Computer-Aided Design,* pp. 463-466, November 1995.

[7] F. Najm, “Probabilistic simulation for reliability analysis of CMOS VLSI circuits,” *IEEE Transactions on Computer-Aided Design of ICs and Systems,* pp. 439-450, volume CAD-9, April 1990.

[8] J. J. Cong and C.-K. Koh, “Simultaneous driver and wire sizing for performance and power optimization,” *IEEE Transactions on VLSI Systems,* pp. 408-425, volume 2, number 4, December 1994.

[9] N. Menezes, R. Baldick and L. T. Pileggi, “A sequential quadratic programming approach to concurrent gate and wire sizing,” *IEEE International Conference on Computer-Aided Design,* pp. 144-151, November 1995.

[10] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli and A. L.Tits, “DELIGHT.SPICE: An optimization-based system for the design of integrated circuits,” *IEEE Transactions on CAD of ICs and Systems,* pp. 501-519, volume CAD-7, number 4, April 1986.

[11] J.-M. Shyu and A. Sangiovanni-Vincentelli, “ECSTASY: a new environment for IC design optimization,” *IEEE International Conference on Computer-Aided Design*, pp. 484-487, November 1988.

[12] C. Visweswariah and R. A. Rohrer, “Piecewise approximate circuit simulation,” in Proceedings of the *IEEE International Conference on Computer-Aided Design,* November 1989.

[13] C. Visweswariah and R. A. Rohrer, “Piecewise approximate circuit simulation,” *IEEE Transactions on CAD of ICs and Systems,* pp. 861-870, July 1991.

[14] A. R. Conn, N. I. M. Gould and Ph. L. Toint, “Global convergence of a class of trust region algorithms for optimization with simple bounds,” *SIAM Journal on Numerical Analysis,* pp. 433-460, volume 25, 1988. See also same journal, pp. 764-767, volume 26, 1989.

[15] A. R. Conn, N. I. M. Gould and Ph. L. Toint, “A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds,” *SIAM Journal on Numerical Analysis,* pp. 545-572, volume 28, number 2, 1991.

[16] A. R. Conn, N. I. M. Gould and Ph. L. Toint, *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A),* volume 17 of Springer Series in Computational Mathematics, Springer Verlag, 1992.

[17] J. J. Moré, “The Levenberg-Marquardt algorithm, implementation and theory,” in Numerical Analysis, G. A. Watson, Editor, *Lecture Notes in mathematics 630*, Springer-Verlag, 1977.

[18] B. A. Murtagh and M.A. Saunders, *MINOS 5.1 User’s Guide*, Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA 94305, December 1983, revised January 1987.

[19] I. Bongartz, A. R. Conn, N. Gould and Ph. L. Toint, “CUTE: constrained and unconstrained testing environment,” *ACM Transactions on Mathematical Software*, pp. 123-160, volume 21, number 1, March 1995.

[20] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quassemzadeh and T. R. Scott, “Algorithms for ASTAP - a network analysis program,” *IEEE Transactions on Circuit Theory,* pp. 628-634, volume CT-20, November 1973.

[21] L. T. Pillage, R. A. Rohrer and C. Visweswariah, *Electronic circuit and system simulation methods*, McGraw Hill, 1995.

[22] T. V. Nguyen, P. Feldmann, S. W. Director and R. A. Rohrer, “SPECS simulation validation with efficient transient sensitivity computation,” in Proceedings of the *IEEE International Conference on Computer-Aided Design,* pp. 252-255, November 1989.

[23] P. Feldmann, T. V. Nguyen, S. W. Director and R. A. Rohrer, “Sensitivity computation in piecewise approximate circuit simulation,” *IEEE Transactions on CAD of ICs and Systems,* pp. 171-183, February 1991.

[24] T. V. Nguyen, *Transient sensitivity computation and applications*, Research Report Number CMUCAD-91-40, Carnegie Mellon University, Pittsburgh, 1991.

[25] S. W. Director and R. A. Rohrer, “The generalized adjoint network and network sensitivities,” *IEEE Transactions on Circuit Theory,* pp. 318-323, volume CT-16, number 3, August 1969.

[26] D. A. Hocevar, P. Yang, T. N. Trick and B. D. Epler, “Transient sensitivity computation for MOSFET circuits,” *IEEE Transactions on CAD of ICs and Systems,* pp. 609-620, volume CAD-4, number 4, October 1985.

[27] J. J. Moré, “Recent developments in algorithms and software for trust region methods,” in A. Bachem, M. Grötschel and B. Korte, editors, *Mathematical Programming: The State of the Art,* pp. 258-287, Springer Verlag, Berlin 1983.

[28] A. R. Conn, N. I. M. Gould and Ph. L. Toint, “On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds,” in D. F. Griffiths and G. A. Watson, editors, *Proceedings of the 14th Biennial Numerical Analysis Conference,* Dundee, pp. 49-68, Longmans, 1992.

[29] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization,* Academic Press, 1981.

[30] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct methods for sparse matrices,* Clarendon Press, Oxford, U.K., 1986.

[31] R. B. Schnabel and E. Eskow, “A new modified Cholesky factorization,” *SIAM Journal on Scientific and Statistical Computing,* pp. 1136-1158, volume 11, 1991.

[32] *Design Entry: Composer Users’ Guide 4.3*, Cadence Design Systems Inc., San Jose, CA, 1994.

[33] M. Rubin, *View: A user tailorable interface for circuit design graphics,* IBM Technical Report, TR-19.90629, August 1990.

[34] M. D. Matson, L. A. Glasser, “Macromodeling and optimization of digital MOS VLSI circuits,” *IEEE Transactions on CAD of ICs and Systems,* pp. 659-678, volume CAD-5, number 4, October 1986.